Clustering algorithms on GPU

Team 18
Deepam Sarmah
2020050
deepam20050@iiitd.ac.in

Introduction

- ➔ Clustering algorithms cluster similar data points based on their characteristics or features.
- ➔ They help segment the data into meaningful groups based on a standard metric (these can be maximum revenue, average life expectancy, etc.) and anomaly detection, where clustering algorithms can help identify outliers or anomalies in the data by identifying data points that do not belong to any cluster (this is helpful in cancer detection)
- ➔ Many clustering algorithms are computationally intensive, and hence the GPU could parallelize these computations owing to their larger memory bandwidth and processing power.

Milestones

S. No.	Milestone	Member
	Mid evaluation	
1	Implement K-Means on CPU	Deepam
2	Implement DBSCAN on CPU	Deepam
	Final evaluation	
3	Implement K-Means on GPU	Deepam
4	Implement DBSCAN on GPU	Deepam
5	Implement Mean-Shift on CPU	Deepam
6	Implement Mean-Shift on GPU	Deepam
7	Compare and analyze GPU code using profiler	Deepam

Clustering Algorithms

1. K-Means Clustering
2. Density-based spatial clustering of applications with noise (DBSCAN)
3. Mean-Shift Clustering (With Gaussian Kernel)

[1] K-Means

- ➔ There are K clusters each of which have a centroid.
- ➔ A data point belongs to the cluster from whose centroid its distance is the least.
- ➔ Lloyd’s algorithm : Widely used algorithm for computing K-Means.

Approach

1. Each thread corresponds to each data point and the work of this thread is to calculate the distance from it to all the K centroids.
2. This thread then assigns the corresponding point to the cluster with the least distance to it.
3. Post that it increments in an array that counts the number of points corresponding to the chosen cluster.
4. It also updates the array consisting of all the points that belong to the chosen cluster.
5. We then recompute the centroids with the help of the 2 arrays mentioned in step 3 and step 4.
6. We repeat this process for the given number of iterations and at the end of this we get our K clusters.

Algorithm Analysis

The following is my analysis for 1 iteration of K-Means:

- Serial runtime $T_s = O(NK)$
- Parallel runtime $T_p = O(K)$
- Cost of Parallel runtime $pT_p = O(pK)$, p is the number of processors
- Total Overhead $T_o = pT_p - T_s = (p - N)K$
- Speedup $S = \frac{T_s}{T_p} = O(N)$
- Efficiency $E = \frac{S}{p} = O(\frac{N}{p})$
- My algorithm will be cost optimal only when $p = N$.

Results - 1(a)

CPU (ms) vs GPU (ms) and Speedup

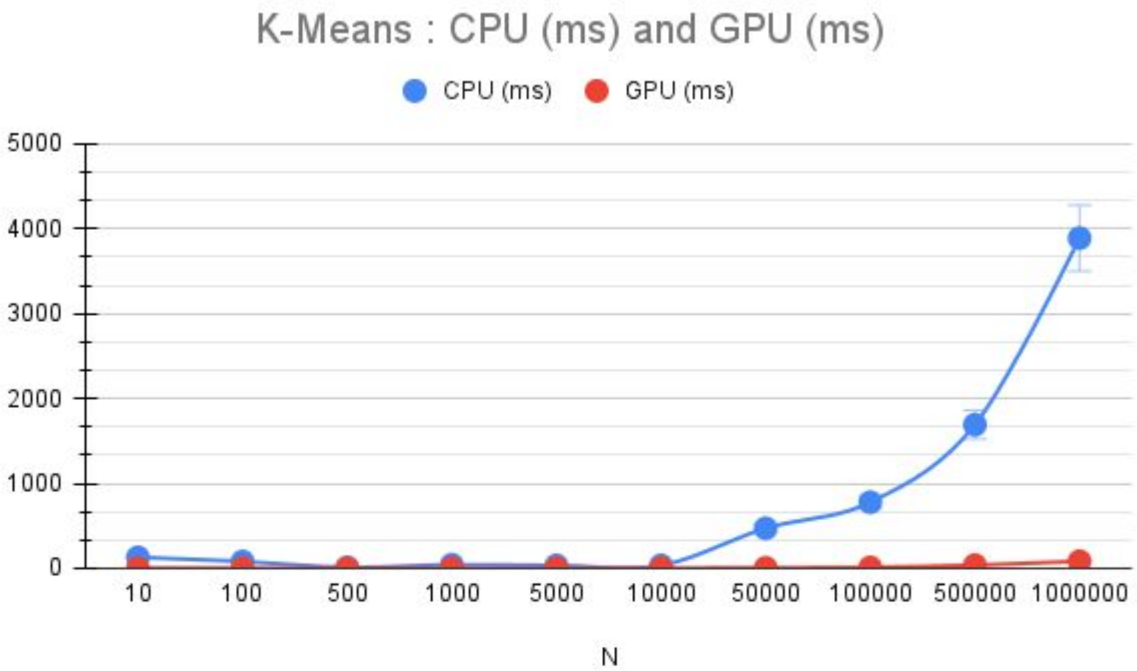
N	CPU (ms)	GPU (ms)	Speedup
10	128.724554	0.687916803	187.1222704
100	80.10202607	0.696799994	114.9569844
500	10.23848236	0.738144004	13.8705758
1000	42.37618176	0.801996803	52.8383425
5000	33.7246001	1.218681622	27.67301934
10000	34.02752513	1.645670414	20.67699877
50000	470.8680856	5.349036694	88.02857646
100000	777.338817	9.996544075	77.76075523
500000	1690.248834	37.23748474	45.39105813
1000000	3887.843754	82.8979599	46.89914877

Results - 1(b)

GPU(ms) vs BLOCK_SIZE for N = 1000000

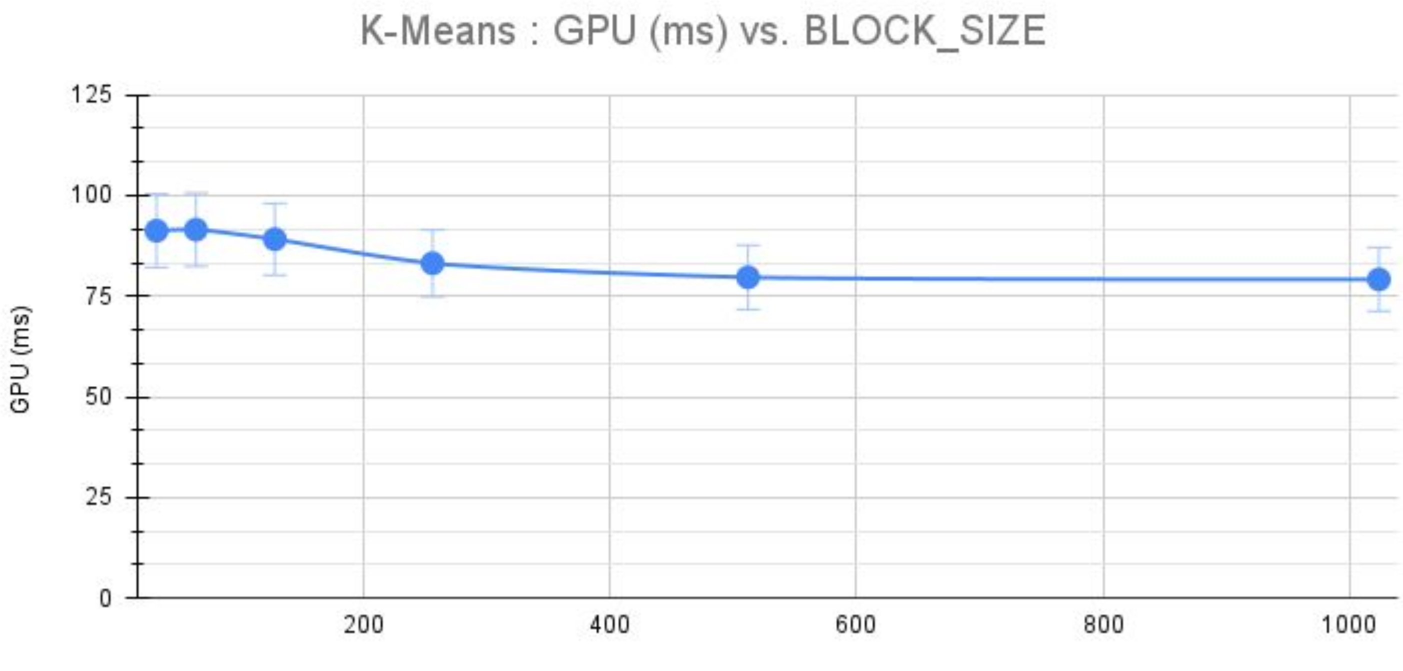
BLOCK_SIZE	GPU (ms)
32	91.26502228
64	91.5320282
128	89.16572571
256	83.18665314
512	79.70175934
1024	79.1830368

Plots - 1(a)



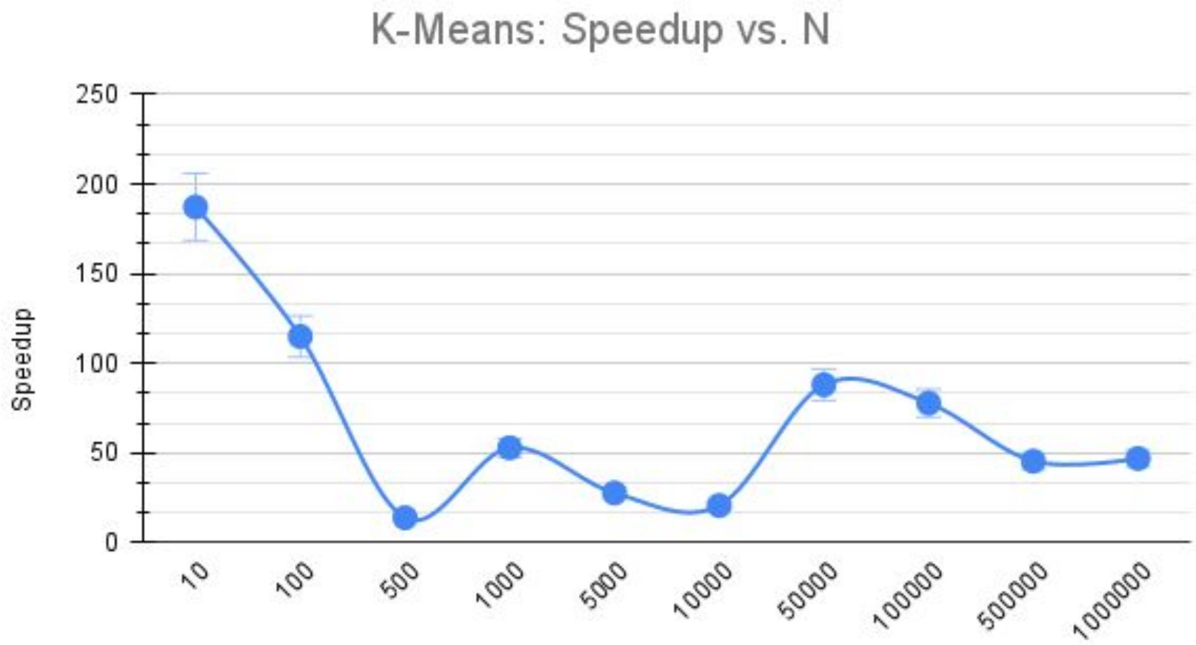
K-Means : CPU (ms) and GPU (ms)
CPU (ms)
GPU (ms)
5000
4000
3000
2000
1000
0
10
100
500
1000
5000
10000
50000
100000
500000
1000000
N

Plots - 1(b)



K-Means : GPU (ms) vs. BLOCK_SIZE
GPU (ms)
125
100
75
50
25
0
200
400
600
800
1000

Plots - 1(c)



K-Means: Speedup vs. N
Speedup
250
200
150
100
50
0
10
100
500
1000
5000
10000
50000
100000
500000
1000000

[2] DBSCAN

➔ There are two main parameters ε - a distance metric and minPts - minimum points metric
➔ Core Points: Points which have at least minPts number of points in a radius of ε.
➔ Border Points: Point which do NOT have minPts number of points in a radius of ε.
➔ Clusters data based on their density.

Approach

- Each thread corresponds to a single data point. The work of this thread is to calculate the distance from it to all the other data points.
- If the number of points which are at distance at most ε is at least minPts then it marks the current point as a core point.
- Then, in another kernel merge all the core points which fall in under distance ε.
- The remaining points are either assigned to a cluster if their distance from its core point is under ε or are marked as noise.
- At the end of all this we get our required clusters.

Algorithm Analysis

- Serial runtime $T_s = O(N \log N)$
- Parallel runtime $T_p = O(N)$
- Cost of Parallel runtime $pT_p = O(pN)$, p is the number of processors
- Total Overhead $T_o = pT_p - T_s = (p - \log N)N$
- Speedup $S = \frac{T_s}{T_p} = O(\log N)$
- Efficiency $E = \frac{S}{p} = O(\frac{\log N}{p})$
- My algorithm will be cost optimal only when $p = \log N$.

Results - 2(a)

CPU (ms) vs GPU (ms) and Speedup

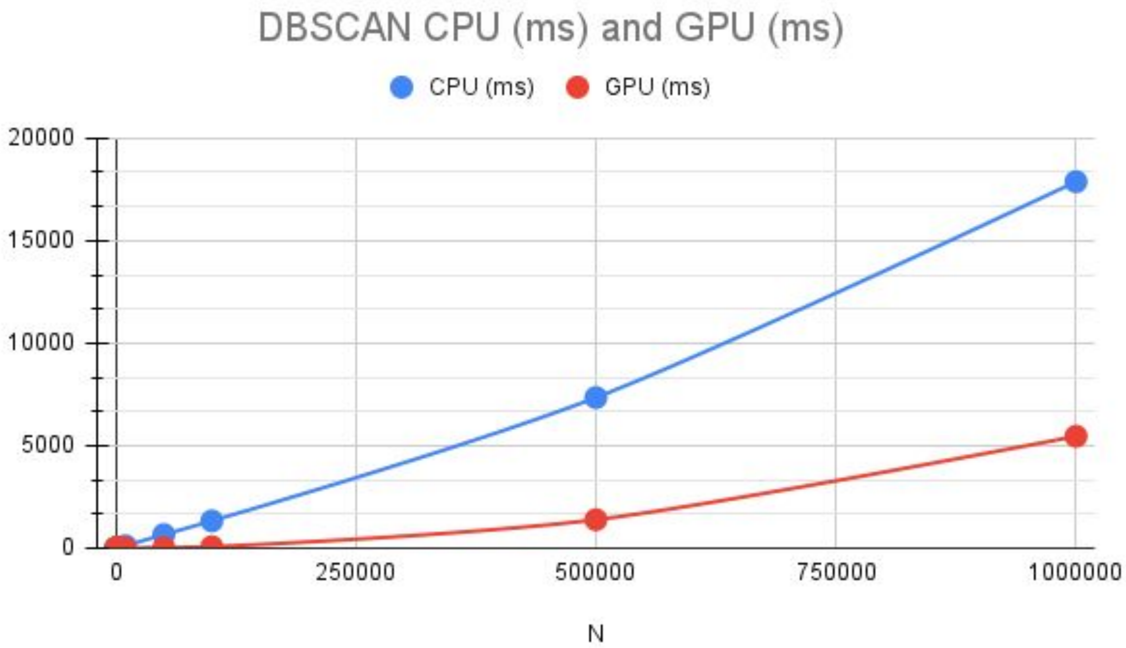
N	CPU (ms)	GPU (ms)	Speedup
10	0.03251452	0.032288	1.00701561
100	0.538092293	0.046566401	11.55537644
500	5.387945659	0.120172802	44.83498403
1000	10.69281772	0.229011199	46.69124377
5000	52.97995079	1.024313617	51.72239235
10000	113.332662	2.051020813	55.25670986
50000	661.0332983	25.33806725	26.08854463
100000	1328.248992	66.61844482	19.93815669
500000	7344.685016	1385.913525	5.299526184
1000000	17875.43273	5455.399707	3.276649501

Results - 2(b)

GPU(ms) vs BLOCK_SIZE for N = 100000

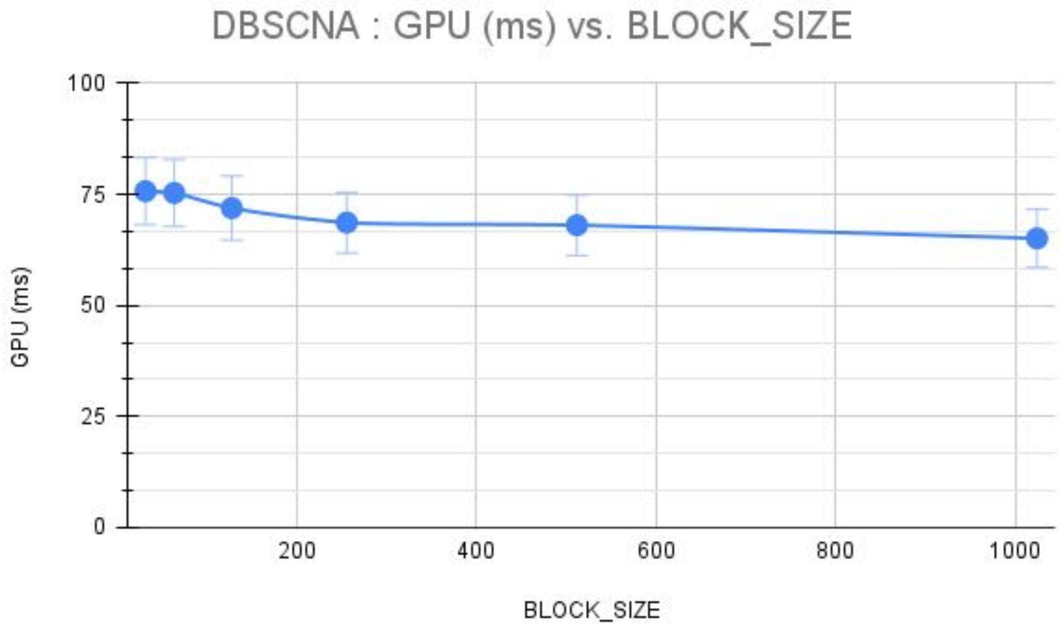
BLOCK_SIZE	GPU (ms)
32	75.7190094
64	75.32182312
128	71.90326691
256	68.65078735
512	68.06060791
1024	65.09737396

Plots - 2(a)

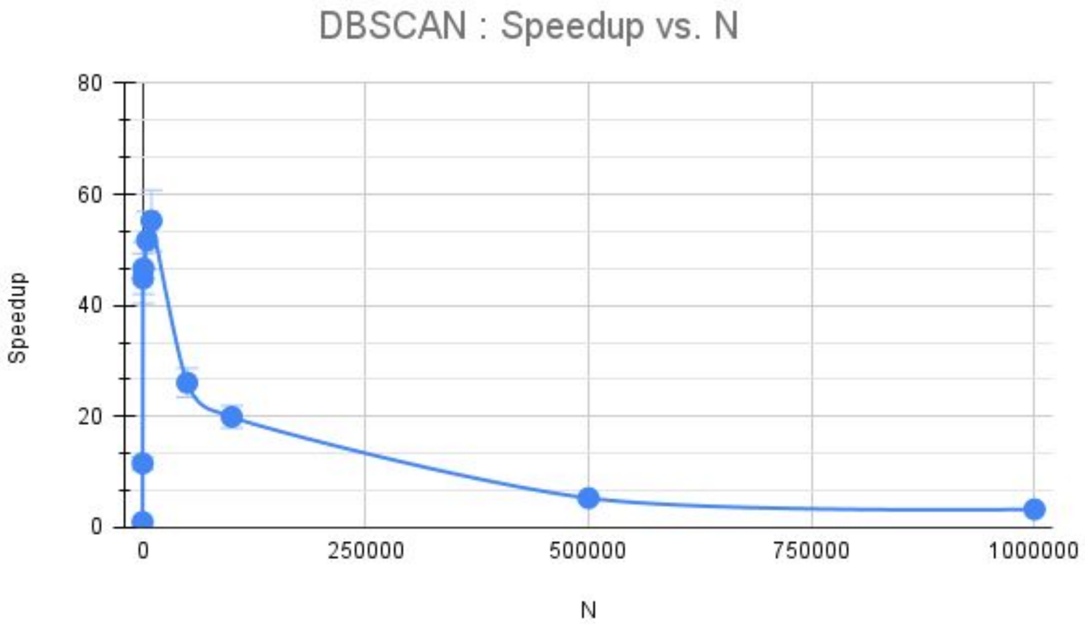


DBSCAN CPU (ms) and GPU (ms)
CPU (ms)
GPU (ms)
20000
15000
10000
5000
0
0
250000
500000
750000
1000000
N

Plots - 2(b)


DBSCNA : GPU (ms) vs. BLOCK_SIZE
GPU (ms)
100
75
50
25
0
200
400
600
800
1000
BLOCK_SIZE

Plots - 2(c)


DBSCAN : Speedup vs. N
Speedup
80
60
40
20
0
0
250000
500000
750000
1000000
N

[3] Mean-Shift Clustering

- For each point we compute the mean by finding the weighted average of the Gaussian kernel value for every other point.
- In the shift stage we shift the respective point to the value of this mean.
- We repeat it for the said number of iterations.
- Points having the same mean after convergence belong to the same cluster.

Approach

- Each thread corresponds to a single point in data. The work of this thread is to compute the weighted gaussian kernel average of all the other points with respect to the chosen point.
- By doing so it computes the mean vector corresponding to the given point.
- Then replace the value of the current point with that of its mean vector.
- We repeat this process for the given number of iterations. Points which have the same mean vector at the end of this belong to the same cluster.

Algorithm Analysis

- Serial runtime $T_s = O(N^2)$
- Parallel runtime $T_p = O(N)$
- Cost of Parallel runtime $pT_p = O(pN)$, p is the number of processors
- Total Overhead $T_o = pT_p - T_s = (p - N)N$
- Speedup $S = \frac{T_s}{T_p} = O(N)$
- Efficiency $E = \frac{S}{p} = O(\frac{1}{p})$
- My parallel algorithm will be cost optimal only when $p = N$.

Results - 3(a)

CPU (ms) vs GPU (ms) and Speedup

N	CPU (ms)	GPU (ms)	Speedup
10	71.61439229	0.047449601	1509.272803
100	50.3189275	0.145407999	346.053366
500	509.2631523	0.633241594	804.2162061
1000	834.4763979	1.694924831	492.3382929
5000	10687.94252	8.660582352	1234.090513
10000	36275.6454	19.73657608	1837.990807
50000	586730	240.4699921	2439.930217
100000	986730	799.0045044	1234.949233
500000	83295000	20897.67695	3985.849728

Results - 3(b)

GPU(ms) vs BLOCK_SIZE for N = 300000

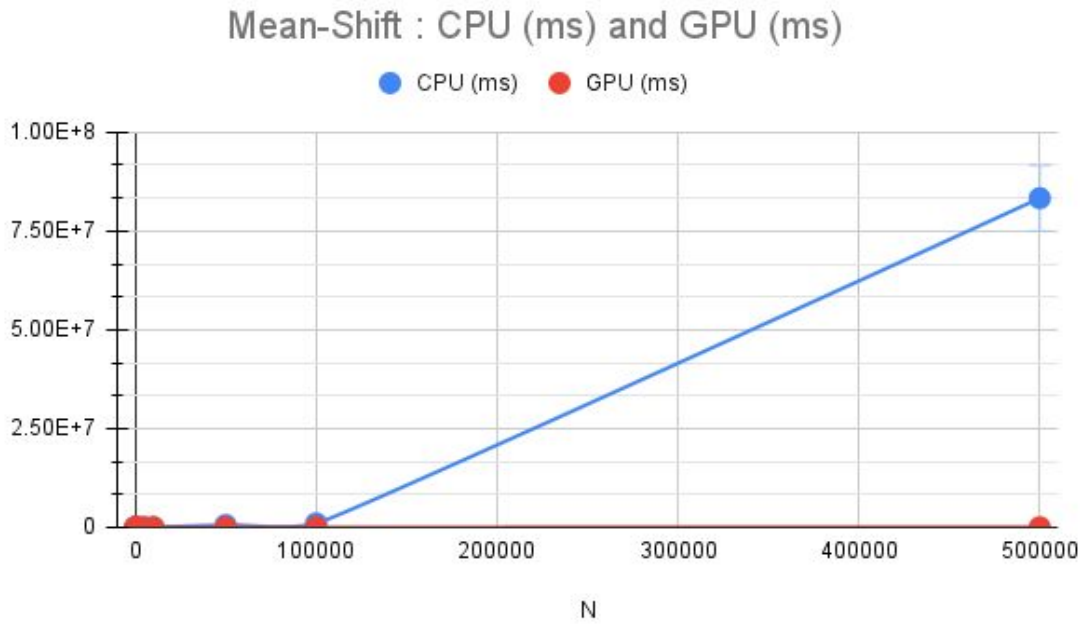
BLOCK_SIZE	GPU (ms)
32	2005.084961
64	2000.337891
128	2003.732544
256	2004.555664
512	2004.428833
1024	2006.924072

Results - 3(c)

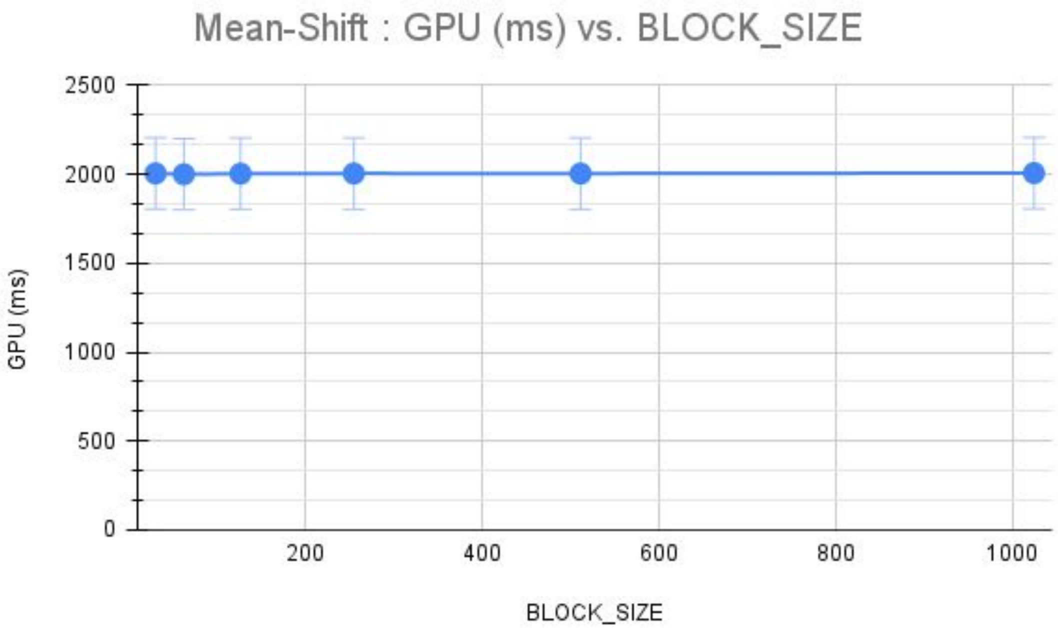
GPU (ms) vs GPU-Shared(ms) and Speedup

N	GPU (ms)	GPU-Shared (ms)	Speedup
10	0.047449601	0.517254388	0.09173358815
100	0.145407999	0.515481591	0.2820818464
500	0.633241594	0.792985618	0.7985536933
1000	1.694924831	1.47681284	1.147691018
5000	8.660582352	7.340646362	1.179811957
10000	19.73657608	14.64811535	1.347379892
50000	240.4699921	186.786911	1.287402799
100000	799.0045044	613.0865356	1.303249147

Plots - 3(a)

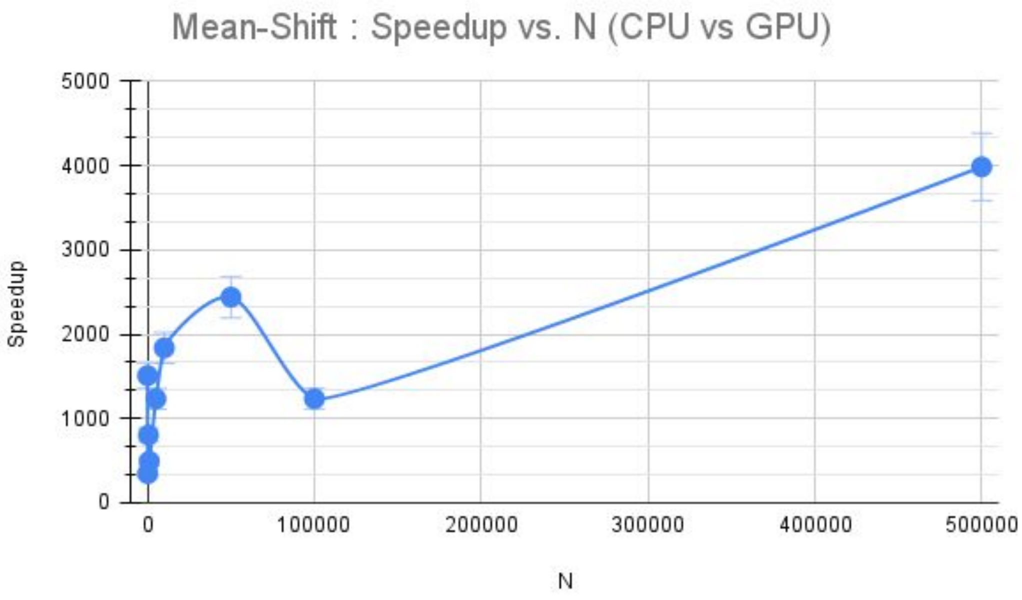

Mean-Shift : CPU (ms) and GPU (ms)
CPU (ms)
GPU (ms)
1.00E+8
7.50E+7
5.00E+7
2.50E+7
0
0
100000
200000
300000
400000
500000
N

Plots - 3(b)



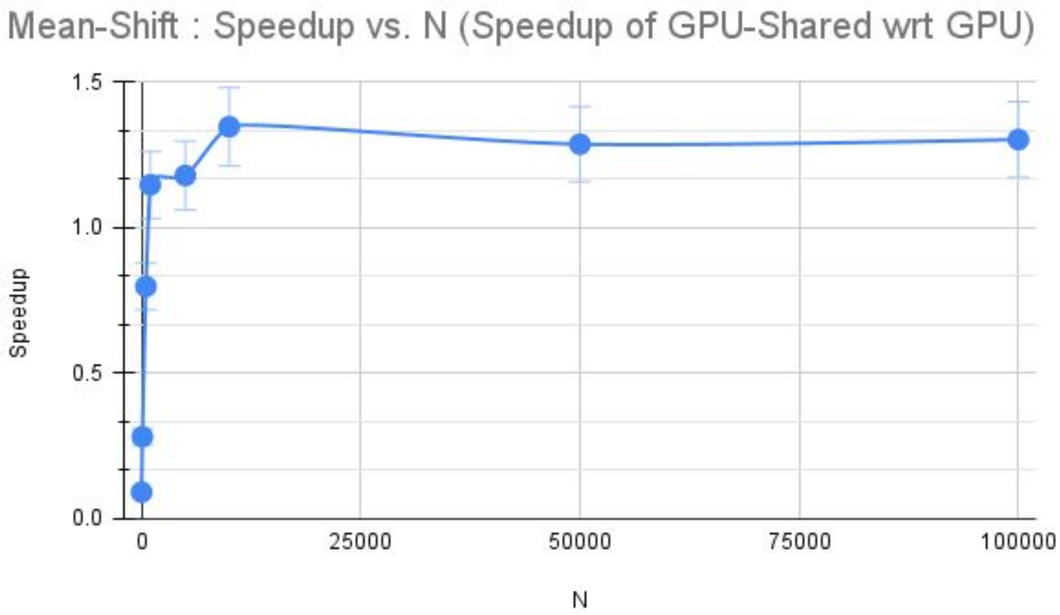
Mean-Shift : GPU (ms) vs. BLOCK_SIZE
2500
2000
1500
1000
500
0
200
400
600
800
1000
GPU (ms)
BLOCK_SIZE

Plots - 3(c)

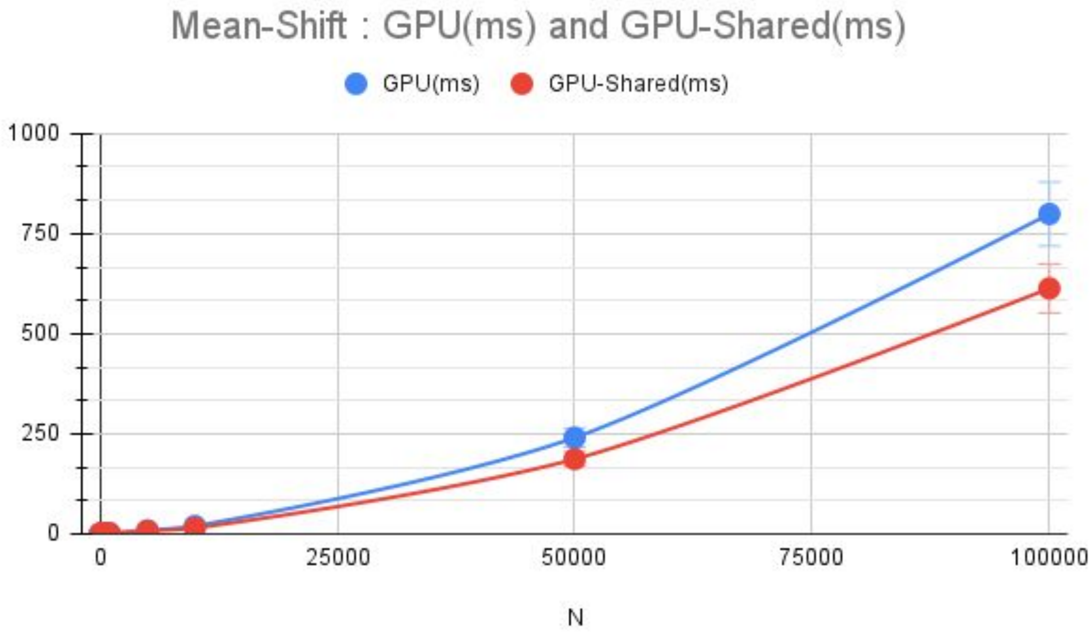


Mean-Shift : Speedup vs. N (CPU vs GPU)
5000
4000
3000
2000
1000
0
Speedup
0
100000
200000
300000
400000
500000
N

Plots - 3(d)


Mean-Shift : Speedup vs. N (Speedup of GPU-Shared wrt GPU)
Speedup
N
1.5
1.0
0.5
0.0
0
25000
50000
75000
100000

Plots - 3(e)


Mean-Shift : GPU(ms) and GPU-Shared(ms)
GPU(ms)
GPU-Shared(ms)
1000
750
500
250
0
0
25000
50000
75000
100000
N

Thank You